# CSP-554 PROJECT REPORT

---

## BIG DATA ANALYSIS OF BRAIN MRI DATASET

Baltej Singh (A20444488)

Abhishek Singh (A20449199)

Yash Agarwal (A20442647)

Suriya Prakaash Sundaram Kasi Thirunavukkarasu (A20449507)

# ABSTRACT

Deep Learning at present is at a stage where it works better than humans on multiple use-cases. Image Classification and Segmentation are the most important problems that humans have been most fascinated with, with the help of computer vision. Treatment is the key stage to improve the quality of life of oncological patients. Magnetic Resonance Imaging (MRI) is the most widely used diagnostic imaging technique to assess such tumors. MRI produces a large amount of data which prevents the Doctors and Radiologists to do the manual segmentation in a reasonable time. Thus, limiting the use of precise quantitative measurements in the clinical practice. The radiologists manually examine these scans to interpret the presence of tumor in the MRI. It is a very unwieldy task to interpret such a huge volume of scans as a daily ongoing process.

Automatic segmentation methods based on deep learning's Convolutional Neural Networks (CNN) can be leveraged to increase the reliability as well as the efficiency with which segmentation is performed on various types of brain tumors. Manually labeled images by domain experts are fed to train the neural network. The idea is to make use of a CNN based on Deep Learning which will be trained on the input data and will learn features present in the MRI without any manual intervention.

# PROJECT DETAILS

- **Project Topic:** Big Data analysis on Brain MRI dataset
- **Application subject area:** Brain MRI Segmentation
- **About Brain MRI Dataset:**

  Brain MRI Dataset is a dataset including MRI scans of two classes, no tumor, or with a tumor. This dataset contains MR images together with manual FLAIR abnormality segmentation masks. We received these images from The Cancer Imaging Archive. It consists of real patient images as well as images created by MICCAI. Each of these folders is then subdivided into high-grade and low-grade images. There are more than 250 patients data included in the Cancer Genome Atlas lower-grade glioma collection with at least FLAIR sequence and genomic cluster data. Tumor genomic cluster and patient data are provided in the data.csv file.

- **Project Goals:**

  - Ingest Brain MRI scan data from the Kaggle website into S3 Bucket.
  - Setting up a SageMaker instance.
  - Downloading the data from the S3 bucket to SageMaker.
  - Perform data profiling & apply necessary transformations.
  - Perform necessary steps for the train and validation split.
  - Training and Prediction of tumor cells from Brain MRI scan reports.
  - Evaluation and Comparing result based on DICE.

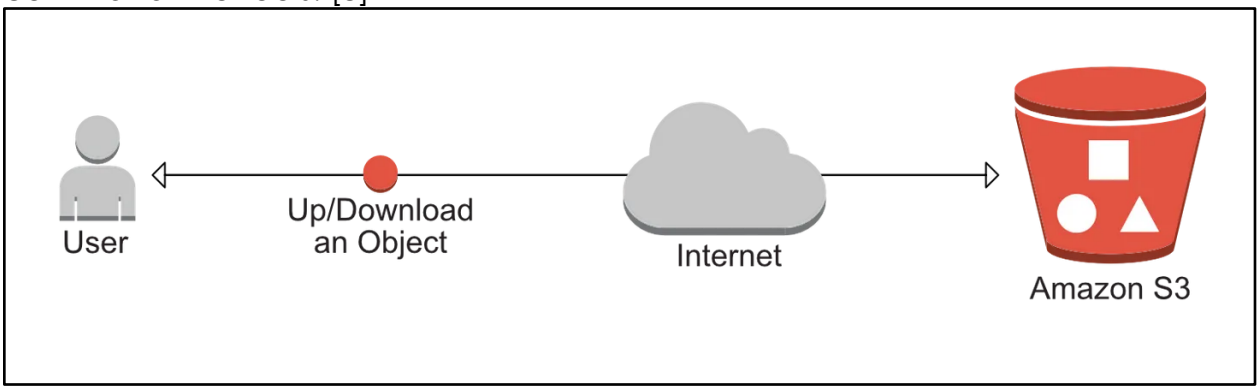
# LITERATURE REVIEW

## AWS

Amazon web services is a platform that offers flexible, reliable, scalable, easy-to-use, and cost-effective cloud computing solutions.
AWS is a comprehensive, easy to use computing platform offered by Amazon. The platform is developed with a combination of infrastructure as a service (IaaS), platform as a service (PaaS), and packaged software as a service (SaaS) offerings. We are using the S3 bucket and Sagemaker for training our Convolution neural net for semantic segmentation on MRI images.

## S3

We will be storing the data for this project in Amazon S3 buckets. Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely tuned access controls to meet your specific business, organizational, and compliance requirements.
S3 Access Points make it easy to manage data access with specific permissions for your applications using a shared data set for this project all the images for the dataset will be stored in S3 buckets. The Sagemaker used for computation will directly stream objects (data) from the s3 bucket into the memory without needing to write it into a file. An S3 protocol URL (like s3://bucket-name/training-data) can be provided as a parameter for any data iterator that takes a file path as input. Also, copying data like artifacts for the trained model from the instance back to another s3 bucket is quite a simple and easy task using the AWS command-line tools. [8]




**S3 Bucket**

**Amazon Sagemaker**

We will be using Sagemaker, another Amazon AWS technology for our project. Sagemaker is a fully managed Machine Learning service that provides developers and data scientists the ability to build, train, and deploy various Machine learning and Deep learning models. Sagemaker provides all of the components used for machine learning in a single toolset so the model can be built, trained, and deployed much faster and with less effort. It provides an integrated Jupyter authoring notebook instance for easy access to the data sources for exploration and analysis, so we do not have to manage servers. Amazon also provides a Python SDK on Sagemaker, using which we can train and deploy models using popular deep learning frameworks, algorithms provided by Amazon, or your algorithms built into Sagemaker-compatible Docker images. The Sagemaker also provides a new capability known as Sagemaker Experiments, that helps organize, track, compare and evaluate machine learning (ML) experiments and model versions by managing iterations and automatically capturing the input parameters, configurations, and results, and storing them as 'experiments'.

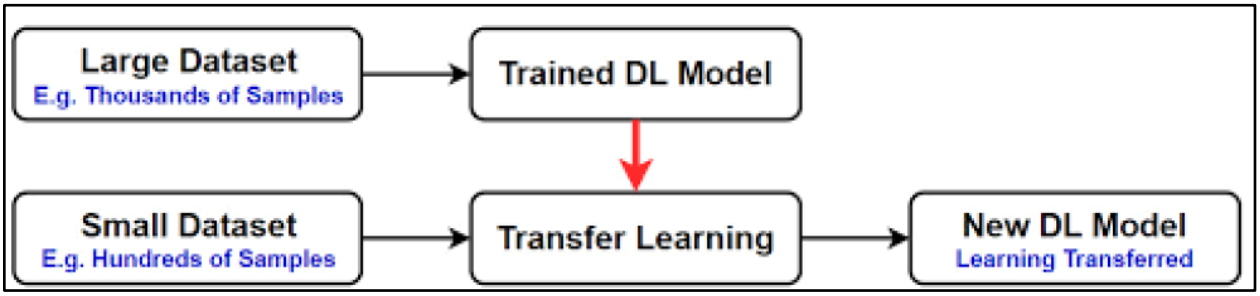
## Semantic Image Segmentation

Image Segmentation is the process of classifying each pixel in an image belonging to a certain class. Semantic Image segmentation is a type of Image Segmentation. It is a computer vision task in which we label specific regions of an image according to what is being shown. It is achieved by labeling each pixel of an image with a corresponding class of what is being represented. Since we are 'predicting' for each pixel, the idea is commonly referred to as dense prediction.

## Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in the skill that they provide on related problems.




## Transfer Learning with Image Data

It is common to perform transfer learning with predictive modeling problems that use image data as input.

This may be a prediction task that takes photographs or video data as input. For these types of problems, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition.

**Pytorch**

PyTorch is an open-source deep learning framework built to be flexible and modular for research, with the stability and support needed for production deployment. PyTorch provides a Python package for high-level features like tensor computation (like NumPy) with strong GPU acceleration and TorchScript for an easy transition between eager mode and graph mode. With the latest release of PyTorch, the framework provides graph-based execution, distributed training, mobile deployment, and quantization. [9]

**fast.ai**

fast.ai is a deep learning library that provides developers and data scientists with high-level components that can quickly and easily provide state-of-the-art results in standard deep learning domains. fast.ai also provides researchers with low-level components that can be mixed and matched to build new approaches. It aims to do both things along with ease of use, flexibility, and performance. fast.ai possesses a carefully layered architecture, which expresses common underlying patterns of many deep learning and data processing techniques in terms of decoupled abstractions. These abstractions are made by leveraging the dynamism of the Python language and the flexibility of the PyTorch library. [19]

| | SageMaker | H2O |
|---|---|---|
| **Meets requirements** | More than H2O | comparatively lesser |
| **Ease of use** | Usage is easier but a bit less than H2O | Very easy to use |
| **Ease of Setup** | Setup is very easy | Setting up is not ease as Sagemaker |
| **Ease of Admin** | Both are almost similar | |
| **Quality of Support** | Both are almost similar | |

| | | |
|---|---|---|
| **Data Ingestion & wrangling** | Lesser than H2O | Comparatively Higher than SageMaker |
| **Language Support** | Python, JavaScript, Ruby, Java, and Go | Java, Scala, R, and Python |
| **Drag and Drop** | Available in SageMaker | Not ease as in SageMaker |
| **Pre-Built Algorithms** | Both have almost the same levels | |
| **Model training** | Comparatively little higher than H2O | Almost up to the level of SageMaker |

**SageMaker vs H2O [21]**

# PROJECT PLAN & MILESTONES

| WBS | Task Name | Resource Names | Notes | Status |
|---|---|---|---|---|
| 0 | Downloading Brain MRI dataset | Baltej, Abhishek, Yash, Suriya | | On Schedule |
| 1 | Install all required tools and utilities | Baltej, Abhishek, Yash | Uploading data to S3, Configuring AWS Sagemaker | Complete |
| 2 | Development and preparation of project | Suriya, Baltej, Abhishek | Architecture Diagram, Task Allocation | Complete |

| | | | | |
|---|---|---|---|---|
| 3 | Ingesting data and reviewing exploratory data | Yash | Upload data from the S3 bucket to Sagemaker. Jupyter Notebook and doing EDA on it. | Complete |
| 4 | Training Model | Suriya, Baltej, | Using the dataset to train the models for semantic segmentation on Brain images | Complete |
| 5 | Evaluating the model | Suriya, Abhishek | Evaluating the accuracy of models. | Complete |
| 6 | Visualization the result | Baltej, Yash | Visualize the results for better insight into how the model is doing. | Complete |
| 7 | Retraining the dataset on different CNN architecture. | Yash, Suriya | Use a dataset to train deeper models. | Complete |
| 8 | Comparing the results of both models and visualizing. | Abhishek | Visualize the results for better insight into how the model is doing and compare the results of both models | Complete |
| 9 | Insights originate from | Baltej, Yash, Abhishek | Look for ways to improve. | Complete |

| | | | | |
|---|---|---|---|---|
| 10 | Produce a Project Report | Suriya, Yash, Baltej | | Complete |

# PROJECT OBJECTIVES & METHODOLOGY

The objective of this project is to increase the reliability as well as the efficiency with which segmentation is performed on various types of brain tumors. To do so, we will build Automatic segmentation methods based on deep learning's Convolutional Neural Networks (CNN). As we know, deep learning problems require a high computation power. Thus, the key to this project is to develop deep learning for semantic segmentation. To do so, the used the following approach:

Building a deep learning model in Amazon SageMaker jupyter notebook: We will build a model for semantic segmentation of images. SageMaker will use Amazon S3 buckets in the notebook. The model will be built using transfer learning and then it will be optimized for our data.

**To validate the results of our project we will test our models on unseen data and calculate the Dice similarity coefficient (DSC) as our metric.**

**Data set description**

The dataset used for this project was sourced from Kaggle, for classification and segmentation of MRI scans of two classes, no tumor, or with a tumor. These images were collected from **The Cancer Imaging Archive**. It consists of real patient images as well as images created by MICCAI. Each of these folders is then subdivided into high-grade and low-grade images. There are more than 250 patients data included in the Cancer Genome Atlas lower-grade glioma collection with at least FLAIR sequence and genomic cluster data. It contains MR images together with manual FLAIR abnormality segmentation masks.

**Training Test set**

**Patient population**

The data used in this study was obtained from The Cancer Genome Atlas (TCGA) and The Cancer Imaging Archive (TCIA). We identified 120 patients from TCGA lower-grade glioma collection1 who had preoperative imaging data available, containing at least a fluid-attenuated inversion recovery (FLAIR) sequence. Ten patients had to be excluded since they did not have genomic cluster information

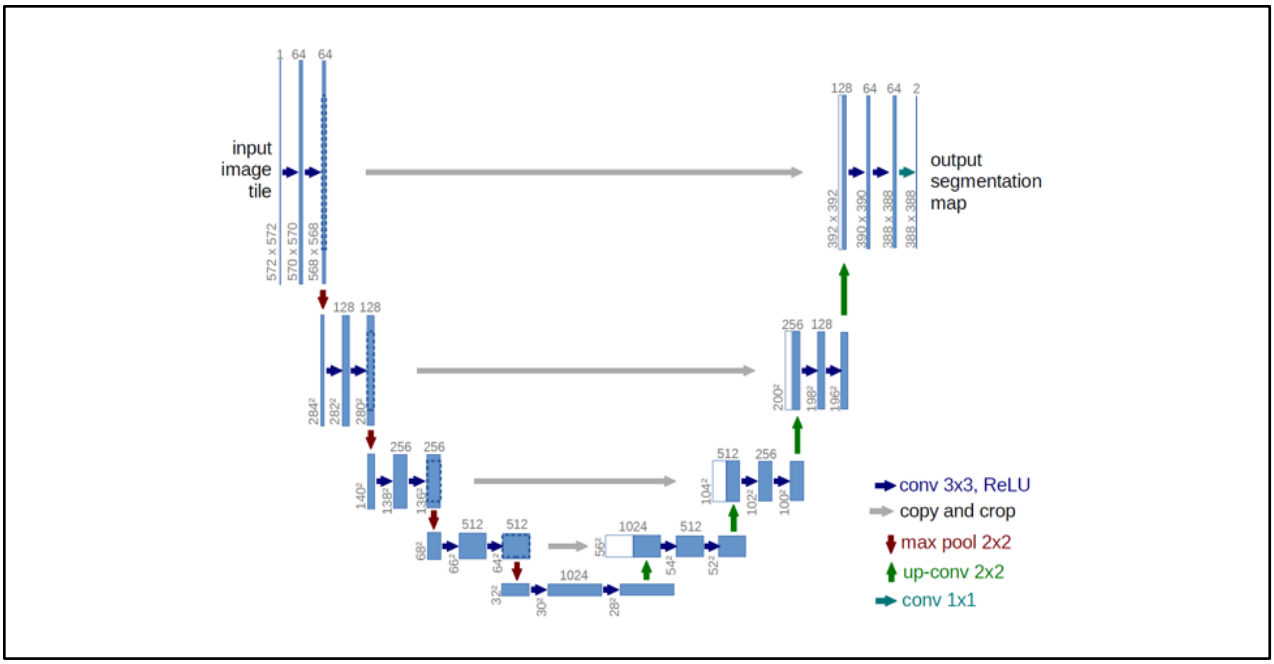
available. The final group of 110 patients was from the following 5 institutions: Thomas Jefferson University (TCGA-CS, 16 patients), Henry Ford Hospital (TCGA-DU, 45 patients), UNC (TCGA-EZ, 1 patient), Case Western (TCGA-FG, 14 patients), Case Western – St. Joseph's (TCGA-HT, 34 patients) from TCGA LGG collection.[15]

## Imaging data

Imaging data was obtained from The Cancer Imaging Archive2 which contains the images corresponding to the TCGA patients and is sponsored by the National Cancer Institute. We used all modalities when available and only FLAIR in case any other modality was missing. There were 101 patients with all sequences available, 9 patients with missing post-contrast sequence, and 6 with missing pre-contrast sequence. The complete list of available sequences for each patient is included in Online Resource 1. The number of slices varied among patients from 20 to 88. To capture the original pattern of tumor growth, we only analyzed preoperative data. The assessment of tumor shape was based on FLAIR abnormality since enhancing tumor in LGG is rare. Dataset of registered images together with manual segmentation masks for each case used in our study is released and made publicly available at the following link: https://kaggle.com/mateuszbuda/lgg-mri-segmentation.[15]

## Segmentation

The main segmentation step was performed using a fully convolutional neural network with the Resnet 18 architecture on the U-Net backbone. It comprises four levels of blocks containing two convolutional layers with ReLU activation function and one max pooling layer in the encoding part and up-convolutional layers instead in the decoding part. Consistent with the U-Net architecture, from the encoding layers we use skip connections to the corresponding layers in the decoding part. They provide a shortcut for gradient flow in shallow layers during the training phase. Manual segmentation served as ground truth for training a model for automatic segmentation. We trained two networks, one for cases with three sequences available (pre-contrast, FLAIR, and post-contrast) and the other that used the only FLAIR. For the second network, instead of missing sequences, we used neighboring FLAIR slices from both sides of a slice of interest as additional channels. Since in this scenario the two sequences, which occupied channel 1 and channel 3 of the input are not available, we filled these channels with neighboring tumor slices to provide additional information to the network.




## Sagemaker Jupyter Notebook

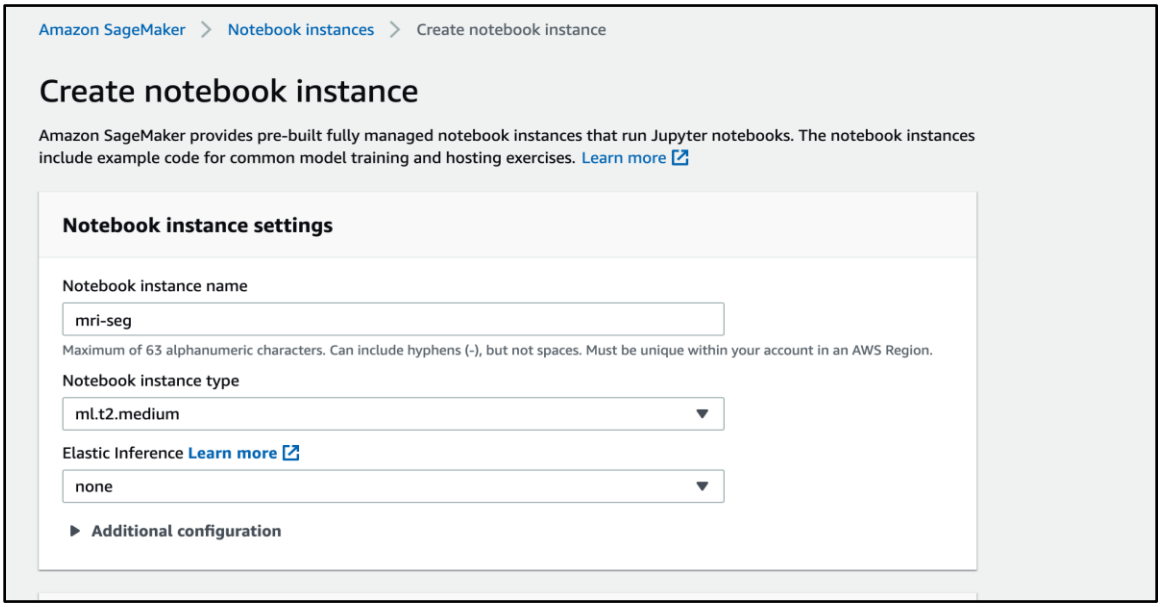
For this experiment, as mentioned before, the main objective was to create a deep learning model for semantic segmentation in a notebook instance using SageMaker. Transfer learning will be used instead of creating our model from scratch. Therefore, we will need to do hyperparameter tuning to customize it to our data set of cancer cells.

### Creating training and validation sets

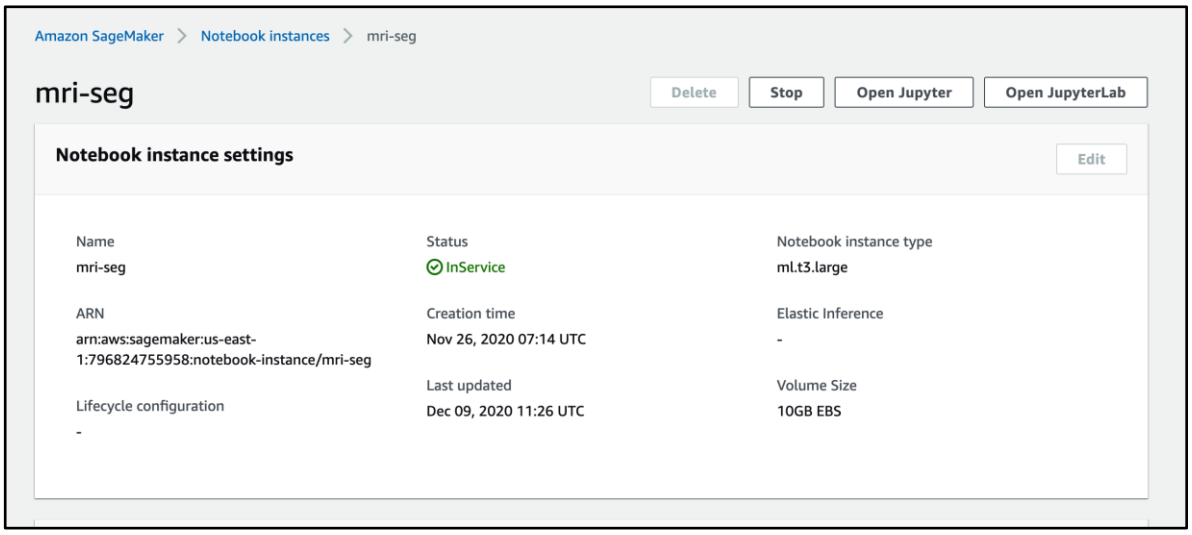
We have 110 patients. We have split the given folders for validation.

```
validation_folders = [
        'TCGA_HT_7694_19950404', 'TCGA_DU_5874_19950510', 'TCGA_DU_7013_19860523',
        'TCGA_HT_8113_19930809', 'TCGA_DU_6399_19830416', 'TCGA_HT_7684_19950816',
        'TCGA_CS_5395_19981004', 'TCGA_FG_6688_20020215', 'TCGA_DU_8165_19970205',
        'TCGA_DU_7019_19940908', 'TCGA_HT_7855_19951020', 'TCGA_DU_A5TT_19980318',
        'TCGA_DU_7300_19910814', 'TCGA_DU_5871_19941206', 'TCGA_DU_5855_19951217']
```

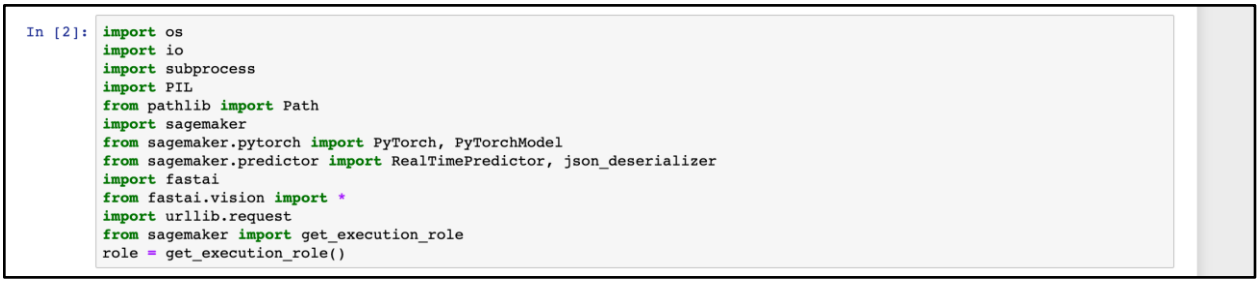
## Create an Amazon SageMaker notebook instance




## Open Jupyter notebook after starting the sagemaker instance




## Importing the required libraries



```
In [2]: import os
        import io
        import subprocess
        import PIL
        from pathlib import Path
        import sagemaker
        from sagemaker.pytorch import PyTorch, PyTorchModel
        from sagemaker.predictor import RealTimePredictor, json_deserializer
        import fastai
        from fastai.vision import *
        import urllib.request
        from sagemaker import get_execution_role
        role = get_execution_role()
```

***from sagemaker import get_execution_role***

The execution role is intended to be available only when running a notebook within SageMaker. When we need to create a sagemaker notebook instance in the AWS console, we need to assign a role.

***from sagemaker.pytorch import PyTorch, PyTorchModel***

Pytorch is an open-source machine-learning framework, for ease of use and pythonic interface. It handles end-to-end training and deployment of custom PyTorch code. This **Estimator** executes a PyTorch script in a managed PyTorch execution environment, within a SageMaker Training Job. The managed PyTorch environment is an Amazon-built Docker container that executes functions defined in the supplied **entry_point** Python script.[20]

***from sagemaker.s3 import S3Downloader***

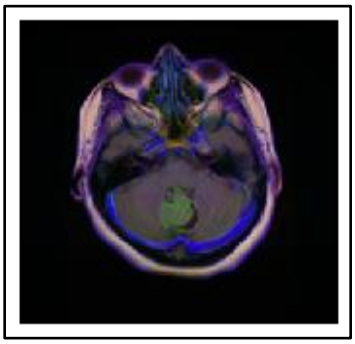
It is a method from SageMaker, it contains static methods for downloading directories or files from S3. It can be used as a utility to view and download the captured data in Amazon S3.

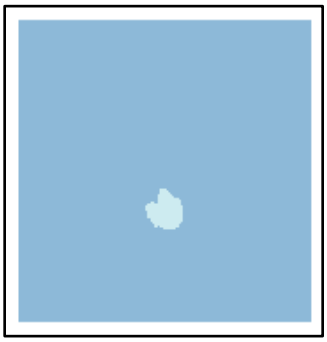
The images that were stored in the S3 bucket will be used for the training and validation of the model. [18]

**Pulling the data from the S3 buckets**

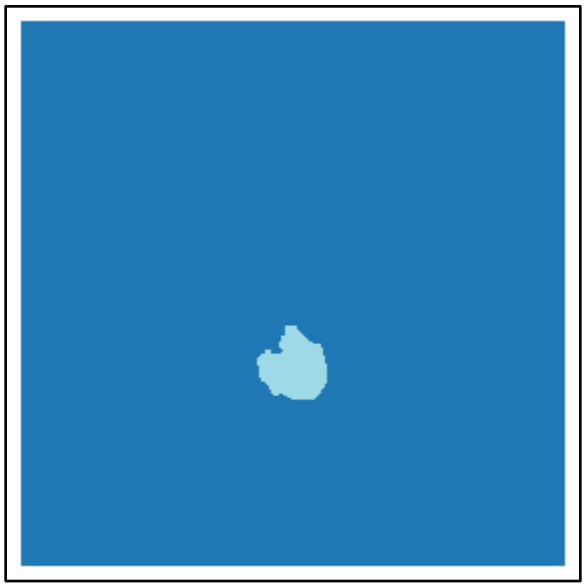

## Visualization of image

```
In [16]: data_path = Path('/home/ec2-user/SageMaker/zh-nbs_2020-11-26/data/')
         small_data_path = data_path/'TCGA_HT_7680_19970202'

In [19]: temp_img_file = small_data_path/'TCGA_HT_7680_19970202_6.tif'
         temp_mask_file = get_y_fn(temp_img_file)
         for f in [temp_img_file, temp_mask_file]:
             print('showing', f)
             if '_mask.tif' in f.name:
                 mask = open_mask(f)
                 print(mask.shape)
                 mask.show()
             else:
                 temp_img = open_image(f)
                 print(temp_img.shape)
                 temp_img.show()

         showing /home/ec2-user/SageMaker/zh-nbs_2020-11-26/data/TCGA_HT_7680_19970202/TCGA_HT_7680_19970202_6.tif
         torch.Size([3, 256, 256])
         showing /home/ec2-user/SageMaker/zh-nbs_2020-11-26/data/TCGA_HT_7680_19970202/TCGA_HT_7680_19970202_6_mask.tif
         torch.Size([1, 256, 256])
```





## Mask image of the tumor



## Preparation of data for training it on the model

```
In [26]: src = (SegmentationItemListWithDiv.from_folder(data_path, recurse=True)
                .filter_by_func(lambda x: not x.name.endswith('_mask.tif'))
                .split_by_valid_func(lambda x: x.parts[-2] in validation_folders)
                .label_from_func(get_y_fn, classes=codes))

In [27]: # start by training on half size images
         data = (src.transform(get_transforms(), size=src_size//2, tfm_y=True)
                 .databunch(bs=bs)
                 .normalize(imagenet_stats))
```

- The target is 0 and 1 label, where 0 is no tumor and 1 is tumor pixel.
- The labeled image with mask ends with the _mask.tif
- We have to spit the training and validation set.
- As we are using transfer learning we are normalizing our dataset with imagenet_stats.

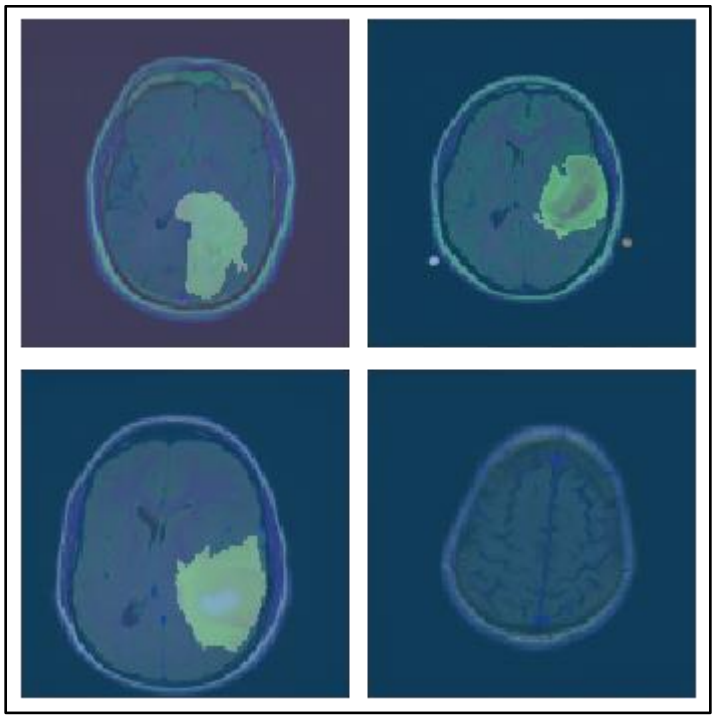
## Training data

```
In [28]: data.train_ds

Out[28]: LabelList (3415 items)
         x: SegmentationItemListWithDiv
         Image (3, 128, 128),Image (3, 128, 128),Image (3, 128, 128),Image (3, 128, 128),Image (3, 128, 128)
         y: SegmentationLabelListWithDiv
         ImageSegment (1, 128, 128),ImageSegment (1, 128, 128),ImageSegment (1, 128, 128),ImageSegment (1, 128, 128),ImageSegm
         ent (1, 128, 128)
         Path: /home/ec2-user/SageMaker/zh-nbs_2020-11-26/data
```

## Validation data

```
]: data.valid_ds

]: LabelList (512 items)
   x: SegmentationItemListWithDiv
   Image (3, 128, 128),Image (3, 128, 128),Image (3, 128, 128),Image (3, 128, 128),Image (3, 128, 128)
   y: SegmentationLabelListWithDiv
   ImageSegment (1, 128, 128),ImageSegment (1, 128, 128),ImageSegment (1, 128, 128),ImageSegment (1, 128, 128),ImageSegm
   ent (1, 128, 128)
   Path: /home/ec2-user/SageMaker/zh-nbs_2020-11-26/data
```

**Images with segmented mask overlapping**



**Loading prepared data on the model for training**

**Resnet 18**

ResNet-18 is a convolutional neural network that is 18 layers deep. We can load a pre-trained version of the network trained on more than a million images from the ImageNet database. The pre-trained network can classify images into 1000 object categories, such as the keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images.

This network has rich features that give a wide range of images. It contains the image input size 0f 224-by-224. The RESNET (Residential Energy Services Network) was found to develop the national standards which are used for the home energy ratings and for creating the market for home energy rating systems. When the Resnet was not found there were many ways to deal with the vanishing gradient issue. The core idea of Resnet is known as identity shortcut connection which skips one or more layers. [14]

**U-Net**

The U-Net is a convolutional network architecture for fast and precise segmentation of images. It is considered the best method for the segmentation

of neuronal structures in electron stacks. It is used to extract the factors in the image. The second part decoder uses transposed convolution to permit localization. It is again an F.C connected layers network.

Started Building the U-Net Model, we have then imported the U-net model being ResNet as a backbone network and loaded weights of image net. We have then defined the input shape that is expected by the base model and the custom layer that takes that base mode input whose output is then passed to the U-Net model. The output of the U-Net model is then passed to other defined ConvNet layers having activation as ReLU. Dice coefficient as the metric, loss function as binray_cross_entropy, and sgd as an optimizer. After defining everything we have compiled the model and fitted the training and validation data to the model.[16]

## Weight Decay

It is a regularization practice by adding a usually L2 norm of weights to the loss function.

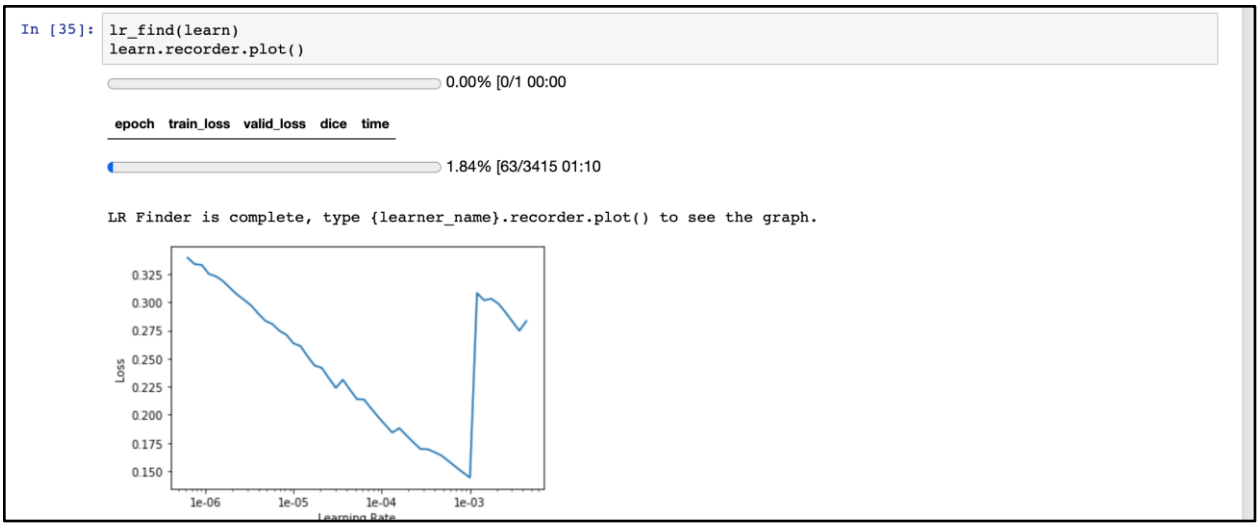
loss = loss + weight decay parameter * L2 norm of the weights

PyTorch applies weight decay to both weights and bias. [17]

```
In [32]: wd=1e-2

In [34]: learn = unet_learner(data, models.resnet18, wd=wd, metrics=dice, path='/home/ec2-user/SageMaker/zh-nbs_2020-11-26/')
         Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /home/ec2-user/.cache/torch/hub/checkpoints/resnet18-5c106cde.pth

         HBox(children=(FloatProgress(value=0.0, max=46827520.0), HTML(value='')))
```

## Learning rate finder

The learning rate is a hyperparameter that controls how much we are adjusting the weights of our network for the loss gradient. It measures how much a model can "learn" from a new mini-batch of training data, meaning how much we update the model weights with information coming from each new mini-batch. The higher the learning rate, the bigger the steps we take along the trajectory to the minimum of the loss function, where the best model parameters are. [12]




## Learning rate

The amount that the weights are updated during training is referred to as the step size or the "learning rate." The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.

A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck.[13]

## Cyclic learning rate

The learning rate is the most important hyper-parameter to tune for training deep neural networks. Cyclical learning rates practically eliminates the need to experimentally find the best values and schedule for global learning rates. Instead of monotonically decreasing the learning rate, this method lets the learning rate cyclically vary between reasonable boundary values. Training with cyclical learning rates instead of fixed values achieves improved classification accuracy without a need to tune and often in fewer iterations. This paper also describes a simple way to estimate "reasonable bounds" -- linearly increasing the learning rate of the network for a few epochs. [11]

We are using a fast.ai library which trains the models using cyclic learning rate methods.

## Training the parameters of the algorithm

Since we are using transfer learning, we will need to customize the parameters of the model to fit our data. The hyperparameters that are specific to the image classification algorithm are:

- **Num_layers**: The number of layers (depth) for the network. We will use 18.
- **Num_classes:** This is the total number of classes. In our case, two, cancer and healthy cells.
- **Epochs:** Number of training epochs.
- **Learning_rate:** The learning rate at which the algorithm gets to a solution. In our case, we decided on a learning rate of 0.0001.
- **Mini_batch_size:** The number of training samples used for each mini-batch.

Since we are using distributed training, the number of samples per batch will be N*mini_batch_size where N is the number of hosts on which the training is running.

## Model Metric -Dice Similarity Coefficient

Dice similarity coefficient is a statistical tool that measures the similarity between two sets of data. The index has become arguably the most broadly used tool in the validation of image segmentation algorithms created with AI but is a much more general concept that can be applied to sets of data for a variety of applications. [10]

The dice coefficient is similar to IoU. They are positively correlated i.e. if model A is better than model B at segmenting an image, then the other will say the same. Like the IoU, they both range from 0 to 1, with 1 signifying the greatest similarity between predicted and truth. [10]

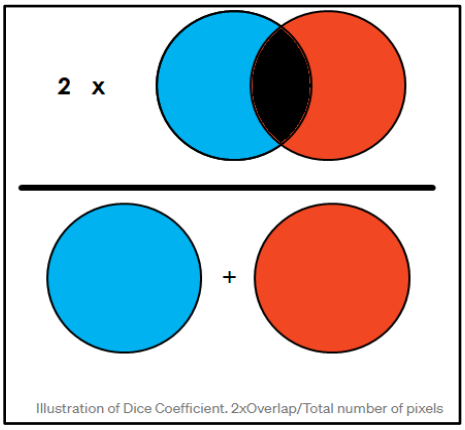


Illustration of Dice Coefficient. 2xOverlap/Total number of pixels

## Training models

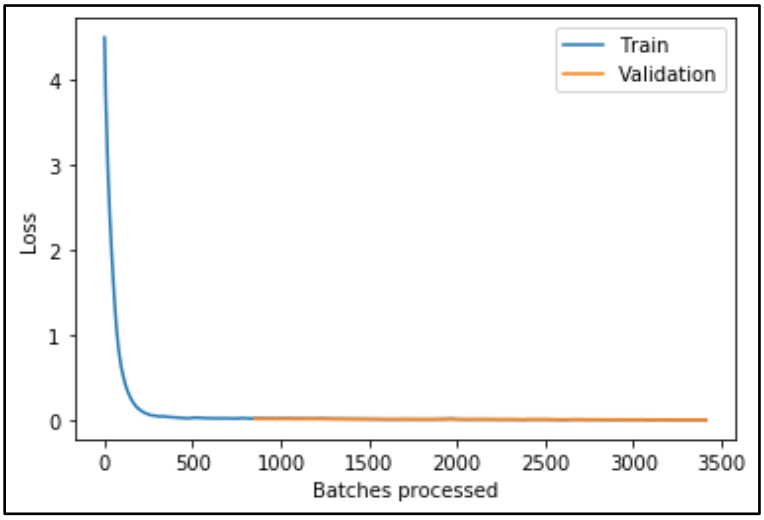
```
In [87]: lr=1e-4

In [88]: learn.fit_one_cycle(4, slice(lr), pct_start=0.9)
```

| epoch | train_loss | valid_loss | dice | time |
|---|---|---|---|---|
| 0 | 0.024652 | 0.019752 | 0.754528 | 54:08 |
| 1 | 0.014357 | 0.016495 | 0.799361 | 52:47 |
| 2 | 0.011672 | 0.012983 | 0.830930 | 50:29 |
| 3 | 0.008976 | 0.009966 | 0.868450 | 50:00 |

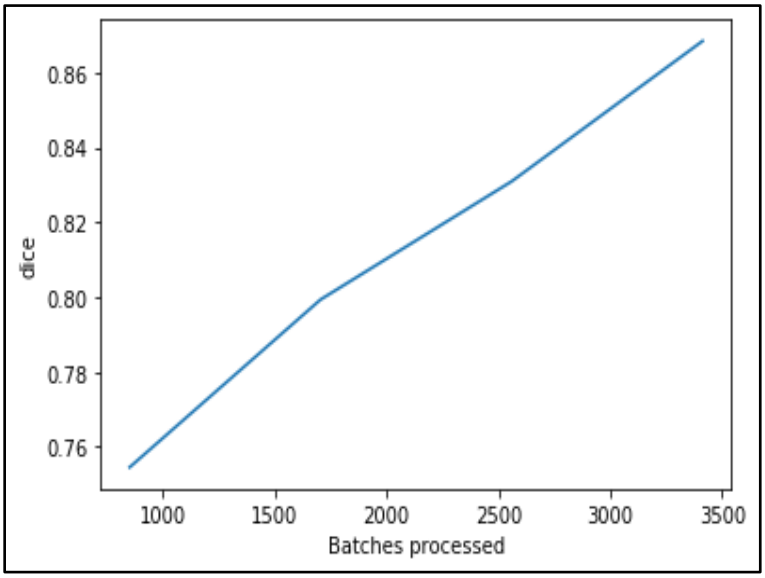
| epoch | train_loss | valid_loss | dice | time |
|---|---|---|---|---|
| 0 | 0.024652 | 0.019752 | 0.754528 | 57:22:00 |
| 1 | 0.014357 | 0.016495 | 0.799361 | 1:01:10 |
| 2 | 0.011672 | 0.012983 | 0.830930 | 1:09:09 |
| 3 | 0.008976 | 0.009966 | 0.868450 | 1:09:59 |

# RESULTS AND CONCLUSION

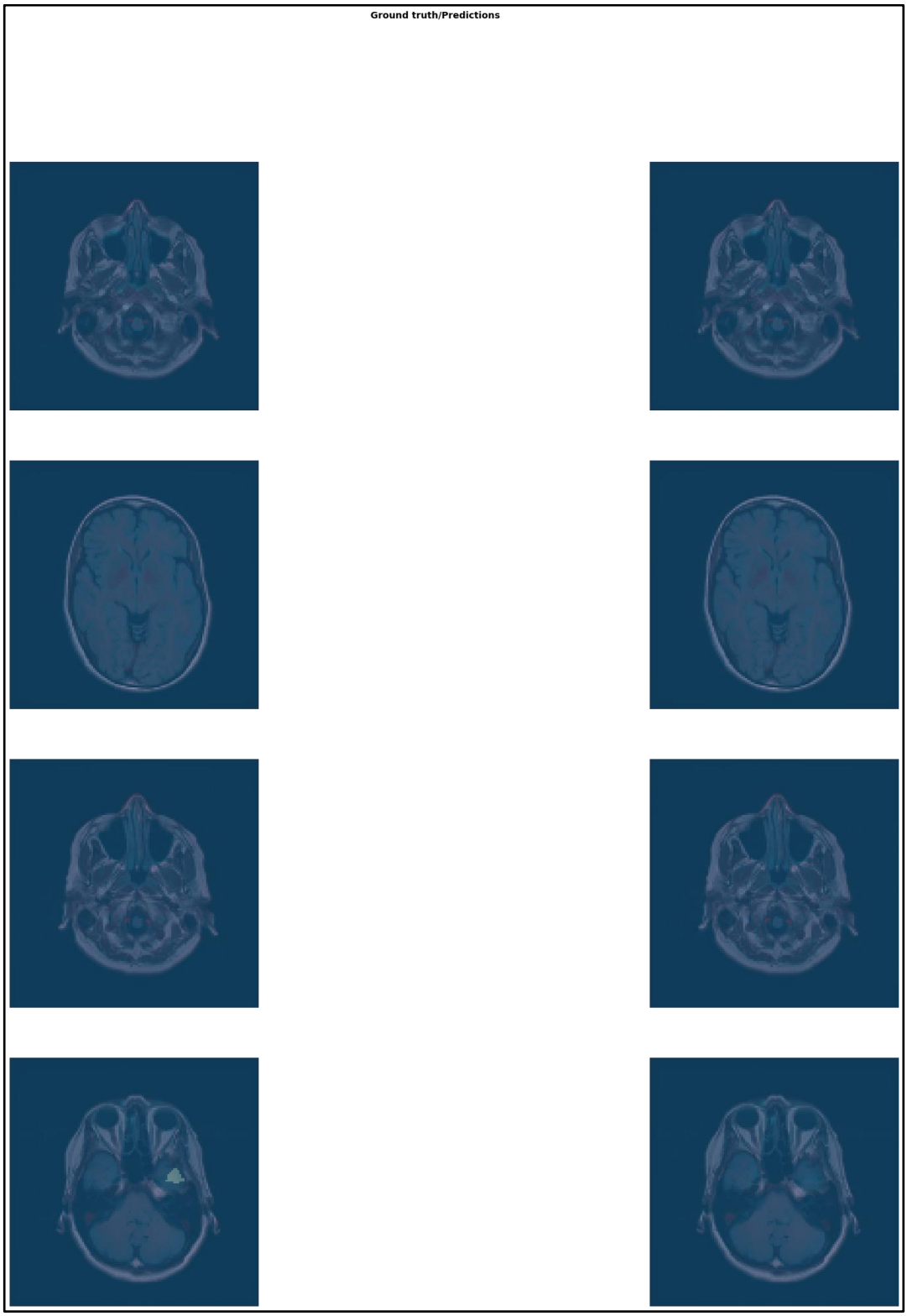
### Training and Validation Loss




### Plot Dice coefficient



## Output results



We have done only 4 epochs due to time constraints but we can see that the loss is decreasing per epoch therefore the dice coefficient will increase and our results will be better.

# PROJECT TAKEAWAYS

In this project we tried to design a hybrid model using ResNet-18, an Eighteen layer Deep Convolutional Neural Network. Working on a Distributed System. There are numerous options to work with for Deep learning, ranging from TensorFlow and Keras to PyTorch and Caffe2. For this project we used fast.ai library for PyTorch for the image processing, which was quite new to us. We chose to use PyTorch over TensorFlow, since as a framework it is more tightly integrated with the Python language and thus feels more native as compared to TensorFlow while working with Deep Learning. What we did learn is that there are many options to work with deep learning to do the image processing even if we were unable to implement the model as we would have wished.

Also, we used Amazon' SageMaker which is a managed Machine Learning service that provides the ability to build, train, and deploy various Machine learning and Deep learning models. It provides an integrated Jupyter authoring notebook instance for easy access to the data sources for exploration and analysis. Also, the integrated Python SDK on Sagemaker, was very helpful for us since it reduced the overhead of installing libraries like PyTorch and fast.ai.

Overall, we learnt about many state-of-the-art technologies while working in this project. We did run into some complications but in the end, we were able to create a model. Although the results were not extremely gratifying, it allowed us to explore very sophisticated technologies and frameworks and learn from them.

# GITHUB REPOSITORY

https://github.com/abhisingh977/brain-mri-segmentation

# REFERENCES


[1] Dataset: Brain MRI segmentation - Kaggle

[2] Sérgio Pereira, Adriano Pinto, Victor Alves, and Carlos A. Silva, "Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images", Published in IEEE Transactions on Medical Imaging vol: 35, issue: 5, DOI: 10.1109/TMI.2016.2538465.

[3] M.A.Balafar, A.R.Ramli, M.I.Saripan, and S.Mashohor, "Review of brain MRI image segmentation methods" in Springer Science Business Media B.V. 2010, DOI 10.1007/s10462-010-9155-0

[4] Jiayun Li, Karthik V. Sarma, King Chung Ho, Arkadiusz Gertych, Beatrice S. Knudsen, and Corey W. Arnold, "A Multi-scale U-Net for Semantic Segmentation of Histological Images from Radical Prostatectomies", at AMIA Symposium April 2018, PMCID: PMC5977596

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, Hartwig Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation" in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 801-818

[6] Rita Yi Man Li, Herru Ching Yu Li, Beiqi Tang, WaiCheung Au, "Fast AI classification for analyzing construction accidents claims" in AICSconf '20: Proceedings of the August 2020 Artificial Intelligence and Complex Systems Conference, DOI: 10.1145/3407703.3407705

[7] Getting Started- Build, Train, and Deploy a Machine Learning Model with Amazon SageMaker

https://aws.amazon.com/getting-started/hands-on/build-train-deploy-machine-learning-model-sagemaker/

[8] Amazon S3, https://aws.amazon.com/s3/

[9] Pytorch, https://ai.facebook.com/tools/pytorch/

[10] Dice similarity coefficient, https://radiopaedia.org/articles/dice-similarity-coefficient

[10] Dice coefficient, https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2

[11] Cyclic learning rate, https://arxiv.org/abs/1506.01186

[12] Learning rate finder, https://blog.dataiku.com/the-learning-rate-finder-technique-how-reliable-is-it

[13] Learning rate, https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/#:~:text=The%20amount%20that%20the%20weights,range%20between%200.0%20and%201.0.

[14]resnet-18, https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

[15] Buda, Mateusz & Saha, Ashirbani & Mazurowski, Maciej. (2019). Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm. Computers in Biology and Medicine. 109. 10.1016/j.compbiomed.2019.05.002.

[16] U-NET, https://pytorch.org/hub/mateuszbuda_brain-segmentation-pytorch_unet/

[17] Weight Decay, https://d2l.ai/chapter_multilayer-perceptrons/weight-decay.html

[18] S3 Downloader, https://sagemaker.readthedocs.io/en/stable/api/utility/s3.html

[19] fast.ai, https://www.fast.ai/

[20] PyTorchModel, https://sagemaker.readthedocs.io/en/stable/frameworks/pytorch/sagemaker.pytorch.html

[21] Amazon SageMaker vs H2O, https://www.g2.com/compare/amazon-sagemaker-vs-h2o